Lifting Freehand Concept Sketches into 3D

Yulia Gryaditskaya, Felix Hähnlein, Chenxi Liu, Alla Sheffer, Adrien Bousseau

Lifting Freehand Concept Sketches into 3D

YULIA GRYADITSKAYA, Université Côte d'Azur, Inria, University of Surrey, CVSSP
FELIX HÄHNLEIN, Université Côte d'Azur, Inria
CHENXI LIU, University of British Columbia
ALLA SHEFFER, University of British Columbia
ADRIEN BOUSSEAU, Université Côte d'Azur, Inria

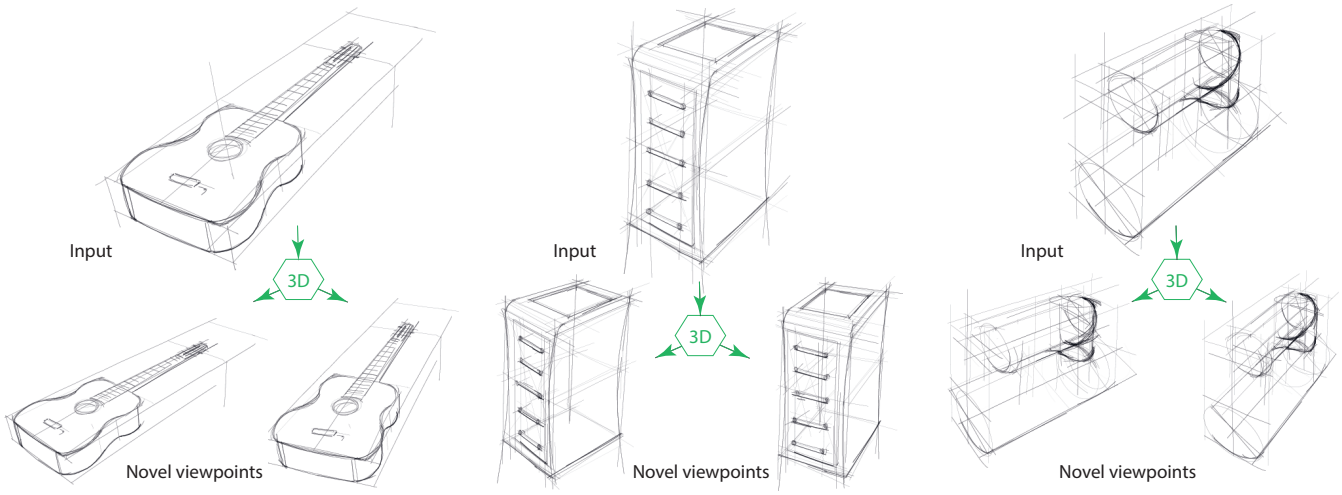


Fig. 1. Given a raw vector 2D design drawing containing both scaffold and surface curves, our algorithm lifts the drawing into 3D by distinguishing 3D intersections from occlusions. The recovered depth enables rendering the drawing from novel viewpoints.

We present the first algorithm capable of automatically lifting real-world, vector-format, industrial design sketches into 3D. Targeting real-world sketches raises numerous challenges due to inaccuracies, use of overdrawn strokes, and construction lines. In particular, while construction lines convey important 3D information, they add significant clutter and introduce multiple accidental 2D intersections. Our algorithm exploits the geometric cues provided by the construction lines and lifts them to 3D by computing their intended 3D intersections and depths. Once lifted to 3D, these lines provide valuable geometric constraints that we leverage to infer the 3D shape of other artist drawn strokes. The core challenge we address is inferring the 3D connectivity of construction and other lines from their 2D projections by separating 2D intersections into 3D intersections and accidental occlusions. We efficiently address this complex combinatorial problem using a dedicated search algorithm that leverages observations about designer drawing preferences, and uses those to explore only the most likely solutions of the 3D intersection detection problem. We demonstrate that our separator outputs are of comparable quality to human annotations, and that the 3D structures we recover enable a range of design editing and visualization applications, including novel view synthesis and 3D-aware scaling of the depicted shape.

Authors' addresses: Yulia Gryaditskaya, Université Côte d'Azur, Inria, University of Surrey, CVSSP, yulia.gryaditskaya@gmail.com; Felix Hähnlein, Université Côte d'Azur, Inria; Chenxi Liu, University of British Columbia; Alla Sheffer, University of British Columbia; Adrien Bousseau, Université Côte d'Azur, Inria.

CCS Concepts: • **Computing methodologies** → **Modeling methodologies**; **Shape modeling**; *Non-photorealistic rendering*; • **Applied computing** → *Computer-aided design.*

Additional Key Words and Phrases: product design, sketching, line drawing, sketch-based modeling, 3D reconstruction

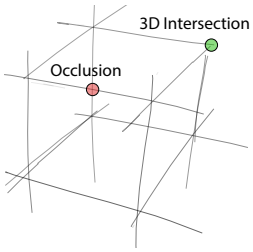
ACM Reference Format:
Yulia Gryaditskaya, Felix Hähnlein, Chenxi Liu, Alla Sheffer, and Adrien Bousseau. 2020. Lifting Freehand Concept Sketches into 3D. *ACM Trans. Graph.* 39, 6, Article 167 (December 2020), 15 pages. https://doi.org/10.1145/3414685.3417851

1 INTRODUCTION

The ubiquity of sketches in design stems from their ability to communicate complex 3D shapes using scattered, swiftly drawn strokes. However, the speed and roughness that makes sketching a convenient medium for designers also makes the computational interpretation of sketches a challenging task, as global 3D understanding needs to emerge from a collection of rough, approximate strokes. Sketch-based modeling research typically sidesteps this challenge by requiring artists to create clean drawings, whose strokes correspond to meaningful surface curves, with clearly annotated connectivity [Lipson and Shpitalni 1996; Liu et al. 2008; Xu et al. 2014] or to use dedicated user interfaces to position strokes in 3D [Bae et al. 2008; Igarashi et al. 1999; Schmidt et al. 2009b]. Unfortunately, both families of methods require extra effort from designers seeking to

lift their sketches to 3D, either by requiring them to learn an entirely new sketching interface, or by asking them to clean-up and annotate their sketches to form what these methods consider as valid inputs. Using general filtering [Simo-Serra et al. 2016] or consolidation [Liu et al. 2018] methods on raw design sketches may reduce the complexity of such clean-up tasks, but does not produce drawings aligned with the expectations of either of the methods above. We thus believe that the additional requirements imposed by these systems prevent their widespread adoption by professional designers. Instead, we argue for *automatic* lifting of raw vector drawings into 3D that would integrate seamlessly into current design practice. Our work is the first attempt to tackle this ambitious goal. Our choice of vector drawings as input is motivated by the ease of use and ubiquity of pen tablets, and by the additional information these tablets record, that enables us to better capture artist intent.

Processing raw real-world product design sketches raises a number of challenges for 3D estimation. In contrast to clean drawings, raw sketches are composed of many overdrawn strokes that do not intersect at precise junctions, and often extend beyond intended end-junctions. They also do not contain stroke connectivity information that would determine which 2D stroke intersections correspond to artist intended 3D curve intersections, and which correspond to accidental occlusions (see inset). As shown by prior work [Lipson and Shpitalni 1996; Xu et al. 2014], accurate stroke connectivity is crucial for correct depth estimation since mistakenly treating occlusions as 3D intersections connects distant parts of the shape, with dramatic consequences on the attempted 3D reconstruction. Automatically discriminating between intended 3D intersections and accidental occlusions is thus critical to our ability to lift the input drawings into 3D space.



We achieve this goal by leveraging the *construction lines* that designers ubiquitously employ when creating even sketchy product drawings (Figure 1a). While these lines often complicate human understanding of drawings by introducing significant visual clutter, they also provide critical geometric cues about the depicted shapes. Correctly classifying intersections across both construction lines and surface strokes is critical to our final goal. Yet, a perceptual study we conducted shows that correctly classifying intersections is extremely challenging even for humans, who only reach an agreement of 89.1% even on simple drawings.

Our framework associates a binary variable with each 2D intersection to indicate if the intersection should be preserved in 3D, and to use these assigned variables to recover a likely 3D interpretation. Our algorithm searches for the assignment of binary values that yields the best 3D shape, as measured via a combination of design-driven metrics. Since the combinatorial nature of this binary assignment problem prevents an exhaustive search of all possible configurations, one of our core technical contributions is an efficient search algorithm that leverages observations about the design sketching process to prioritize more likely assignments.

We make the computation tractable using the following core components. We observe that while designers ubiquitously employ so-called straight *scaffold lines* to support the construction of smooth curves, curves are practically never drawn as a support for straight-line scaffolds. Following this observation, we first process straight lines present in the artist drawings, detecting 3D intersections between them and imbuing them with depth. We exploit the fact that these scaffolds are dominated by axis-aligned strokes; detecting and enforcing axis-alignment dramatically reduces the solution space we operate on for these strokes. We then leverage these scaffolds to lift the remaining curves into 3D, by identifying scaffold-curve and curve-curve intersections and by employing both positional and tangential alignment cues at these intersections. We further constrain the reconstruction by assuming planar curves, which is true for a large majority of the man-made objects we considered.

Our second, most important, observation is that human observers can easily lift into 3D not only the final completed drawings but also the intermediate ones, created incrementally as the artist proceeds; moreover, these interpretations remain consistent throughout. We speculate that a core reason for the expressivity of the intermediate drawings, is that artists, by design, employ an incremental construction strategy, and use existing strokes as anchors when introducing new drawing strokes. Existing strokes thus form an informative context that suggests the 3D configuration of the new strokes. This observation implies that rather than considering the entire drawing at once during 3D intersection detection, we can instead follow the drawing order and build the 3D curve network progressively by solving local rather than global assignment problems.

Using such a progressive approach, we process strokes using the drawing order. We solve a local binary assignment problem for each stroke when the local context is sufficient to allow an unambiguous depth interpretation. We robustly address ambiguous intermediate states by delaying assignment until a dominant interpretation emerges. We support this delayed decision strategy by exploring several concurrent interpretations in parallel as the method progresses along the drawing sequence. We prevent combinatorial explosion by stopping this exploration as early as possible, *i.e*, as soon as the corresponding ambiguous strokes can be assigned with confidence. In our experiments we were able to assign labels to stroke intersections right away 57% of the time, and were able to generate unambiguous interpretations for the remaining strokes after processing just a few additional strokes (median of 7 strokes).

We demonstrate the effectiveness of our approach by lifting into 3D a diverse collection of freehand industrial design sketches drawn on a pen tablet by several professional designers. We showcase the applicability of our method by demonstrating a range of applications that benefit from the provided 3D estimates, including novel view synthesis and 3D-aware shape editing. We also performed a perceptual study that reveals that our algorithm takes similar decisions as human observers in distinguishing 3D intersections from occlusions, and that it makes more errors in situations that are also ambiguous to humans.

In summary, we make the following contributions:

- The formulation of depth estimation from raw industrial design drawings as the problem of assigning binary activation values to intersections.
- An efficient search algorithm that exploits stroke drawing order to identify 3D intersections.

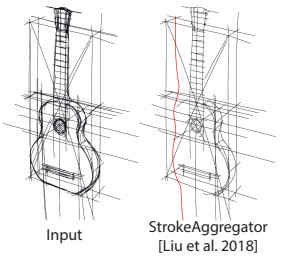
- Taken together, these ingredients allow the first algorithm to automatically estimate stroke depth in raw industrial design sketches that contain scaffold lines.

As pointed out by prior work [Xu et al. 2014], artist sketches are inherently inexact; artists often fail to accurately account for perspective distortion and to depict symmetric, or parallel features [Schmidt et al. 2009a]. Our framework is not designed for correcting such artifacts and does not rely on high-level perceptual cues such as symmetries, frequently leveraged by human observers. Consequently, while our outputs are accurate enough to facilitate a range of high-level image processing tasks, additional work is needed for tasks such as modeling, which require generating outputs whose 2D projection deviates from the drawn input.

2 RELATED WORK

A vast body of work aims to recover 3D depth information from 2D sketches [Bonnici et al. 2019]. Artists employ different domain-specific strategies when depicting different types of shapes and targeting different audiences. Our work targets *concept sketches* that designers draw to best explain a shape they have in mind [Eissen and Steur 2008, 2011]. These sketches ubiquitously employ construction lines that help designers draw 3D surfaces in perspective and help observers interpret the drawn shapes.

Sketch consolidation. Sketch vectorization and consolidation algorithms aim to automatically extract clean vector curves from raw free-hand sketches [Bessmeltsev and Solomon 2018; Favreau et al. 2016; Liu et al. 2018, 2015]. Unfortunately, state-of-the-art methods still struggle with the complex drawings we are targeting. In particular, existing methods were not designed to handle scaffold lines and often group them together with adjacent curves, leading to loss of important context for 3D estimation, as shown in the inset. More importantly, these methods focus on clustering strokes into long curves rather than on recovering the 3D curve network connectivity. When forming the clean output curves, they frequently fail to preserve user intended 2D connectivity making the results ill-suited for subsequent 3D estimation [Liu et al. 2018]. Our initial sketch analysis step utilizes cues highlighted by these approaches to reduce the complexity of our subsequent 3D estimation process (Section 4).



Augmenting Sketches with 3D Effects. We follow a line of work that aims to imbue drawings with depth or normals to facilitate 3D effects such as shading [Shao et al. 2011; Sýkora et al. 2014] and view changes [Liu et al. 2013]. Like these methods we keep the 2D coordinates of all input strokes in place and compute the best 3D estimates given these fixed 2D positions. This approach contrasts with recent 3D reconstruction methods which change the 2D stroke geometry to overcome artist inaccuracies [Xu et al. 2014]. This corrective approach is not suitable for sketch augmentation applications where users expect the 2D strokes to remain in place. We use the computed depth to support design exploration and communication by allowing designers to visualize their sketches from novel viewpoints, and to modify the depicted shape while maintaining consistent perspective. These applications were inspired by related work in computational photography, where approximate depth is commonly used to synthesize nearby viewpoints that bring life to still photographs [Hoiem et al. 2005; Niklaus et al. 2019].

Sketch-Based Modeling. A number of sketch-based modeling systems propose interactive workflows where users create 3D shapes using dedicated interfaces [Bae et al. 2008; Dorsey et al. 2007; Igarashi et al. 1999; Nealen et al. 2007; Olivier et al. 2019; Orbay and Kara 2012; Paczkowski et al. 2011; Zeleznik et al. 1996]. Users of these systems typically alternate between 2D sketching and 3D camera rotation to incrementally create the desired shape. Alternatively, single-view systems leverage user annotations to align parametric shapes with image contours [Chen et al. 2013; Shao et al. 2013; Shtof et al. 2013] or to obtain constraints on the length, orientation or depth ordering of object parts [Gingold et al. 2009; Sýkora et al. 2014]. In contrast to this body of work, we propose a method capable of lifting *real-world* product design sketches to 3D *automatically*. Users of our approach do not need to learn a new interface or specific sketching gestures and annotations. Instead, we leverage geometric cues provided by scaffolds, a drawing element that product designers already ubiquitously use to construct their drawings.

Interactive Scaffolding. Our approach is inspired by the work of Schmidt et al. [2009b], who proposed an interactive scaffold-based sketch-based modeling interface. Their method targets clean drawings where each stroke conveys a complete, meaningful geometric curve. It recovers the 3D shape of each line or curve as soon as it is drawn by the user by inferring a set of positional and alignment constraints between the new curve and already-created scaffold lines. Since this inference occurs within an interactive system, users are expected to correct erroneous solutions by re-drawing the curve, by providing additional guidelines, or by disabling conflicting constraints. As a result, Schmidt et al. report that users needed extended training to learn how to construct complex drawings with their dedicated interface, and that users spent more time orbiting around the 3D sketch to verify the inferred geometry than drawing it, spending between 45 to 140 minutes to complete a single sketch. In contrast, we aim for an automatic method capable of reconstructing complete real-world sketches composed of dozens or even hundreds of raw, overdrawn strokes, a much harder problem that requires delaying decision until enough context is available. Our automatic approach allows designers to follow their usual drawing practice, with each sketch on average being drawn in around 15 minutes [Gryaditskaya et al. 2019].

Single View Drawing Interpretation. Our work is closer in spirit to automatic single-view drawing interpretation methods. Early work focused on drawings of polyhedral shapes that are dominated by planar, mutually orthogonal faces [Lipson and Shpitalni 1996; Liu et al. 2008]. Recent methods reconstruct drawings of curved surfaces by exploiting symmetry [Chen et al. 2008; Cordier et al. 2013] or orthogonality of designer-drawn curvature lines [Li et al. 2017; Shao et al. 2011; Xu et al. 2014], or by first approximating curved

patches with polygons [Wang et al. 2009]. These methods target clean drawings and expect users to annotate all 2D intersections that depict accidental occlusions rather than intersections. Processing raw drawings using these systems requires users to manually retrace and annotate them (see [Xu et al. 2014], Fig. 16). Our inputs are distinctly different – they contain hundreds of raw strokes and have zero connectivity annotation. By automatically discriminating stroke intersections from occlusions based on 3D inference, our method is the first to reconstruct 3D strokes from raw product design sketches.

Deep learning has recently shown success in inferring 3D information from bitmap line drawings [Delanoy et al. 2018; Li et al. 2018; Lun et al. 2017], alleviating the need for clean vector curves as input. However, such systems require large amounts of paired sketches and 3D models for training. While non-photorealistic rendering algorithms can be used to generate drawings for this purpose, there is a significant domain gap between such synthetic data and real-world product design sketches. We refer the interested reader to the recent study by Gryaditskaya et al. [2019], who showed that a state-of-the-art deep network trained on synthetic drawings get confused by the many hidden and construction lines present in the sketches we target. The problem of parsing and reconstructing 3D wireframes has also been considered in computer vision, taking as input one or multiple photographs [Huang et al. 2018; Usumezbas et al. 2016; Zhou et al. 2019], but these methods do not face the challenge of dealing with occlusion between lines since photographs only capture visible surfaces.

3 PROBLEM FORMULATION

Our method takes as input a single vector-format line drawing of an object captured on a pen tablet. This drawing is composed of a series of time-stamped strokes, represented as polylines. Our ultimate goal is to estimate the artist intended 3D coordinates of all input strokes. We achieve this goal by leveraging the presence of straight-line *scaffolds*, that designers draw as a support to construct complex 3D shapes, as illustrated in Figure 2.

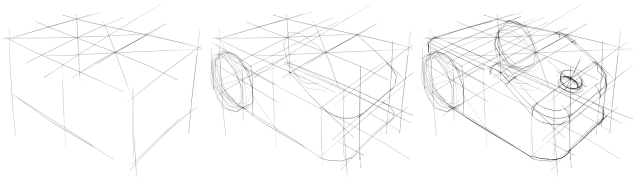


Fig. 2. Three steps of a typical sketching sequence, showing how designers progressively construct 3D shapes using straight-line scaffolds.

3.1 Principles of scaffold-based design sketching

By studying the industrial design sketching literature [Eissen and Steur 2008; Robertson and Bertling 2013] as well as real-world sketching sequences [Gryaditskaya et al. 2019], we identified several strongly repeated patterns in the way designers construct and utilize scaffold strokes.

Ordering. Designers draw strokes using a meaningful construction order, starting with major scaffold lines before adding more fine scaffold details, and curves.

Axis-alignment. To simplify perspective drawing, scaffolds are typically dominated by straight axis-aligned strokes that converge towards two or three vanishing points.

Dense 3D connectivity. Designers exploit existing 3D intersections as anchor points to draw new lines. Consequently, observers do not require the drawing to be complete to perceive its 3D structure.

Embedding of lines and curves. Scaffolds are ultimately drawn to support space curves, and uniquely define their geometry. They are designed so that curves pass through scaffold intersections, are typically tangential to scaffold lines, and lie in planes spanned by scaffold lines. Complex non-planar curves are often broken into planar segments to best benefit from scaffold support.

Bounding volume. Scaffolds are expected to contain, or bound the depicted shapes. Designers commonly draw both the target shape and the scaffold from outside in, starting with a bounding object and then refining it.

Ellipses. Cylinders are frequently used to depict both real cylindrical object features and as a scaffold for more complex geometries. In perspective drawings they are depicted as ellipses.

Minimal foreshortening. Designers favor informative viewpoints that show all sides of the shape with small and evenly-distributed foreshortening, as already noticed by Xu et al. [2014].

Clarity. While scaffolds serve the construction of complex drawings, they should not obscure its content. Designers manage visual clutter by avoiding extending strokes beyond their intended endpoint intersections.

We conjecture that, by providing multiple geometric constraints for designers to construct their drawings, scaffold lines also provide multiple constraints for our algorithm to infer the 3D shape represented by these drawings. However, while most lines in a scaffold satisfy the above characteristics, many do not, due to the approximate nature of freehand sketching. A key challenge we face is to balance the various geometric cues available and select a plausible solution even in the presence of ambiguity. In particular, we need to disambiguate between the intended connectivity of the 3D stroke network and the accidental occlusions due to projection on the drawing canvas.

3.2 Depth prediction given known 3D connectivity

The problem of recovering the depth of a network of 2D strokes when its 3D connectivity is known has been well investigated [Kang et al. 2004; Lipson and Shpitalni 1996; Tian et al. 2009]. In the following we briefly describe the general approach for doing so. Let us denote by $\mathcal{S}$ the input strokes.

Let us also assume that straight axis-aligned strokes $\mathcal{S}_a$ have been identified as such, along with straight non axis-aligned strokes $\mathcal{S}_n$, and planar curved strokes $\mathcal{S}_c$, such that $\mathcal{S} = \{\mathcal{S}_a, \mathcal{S}_n, \mathcal{S}_c\}$. The mutual orthogonality of the axis-aligned strokes can be exploited

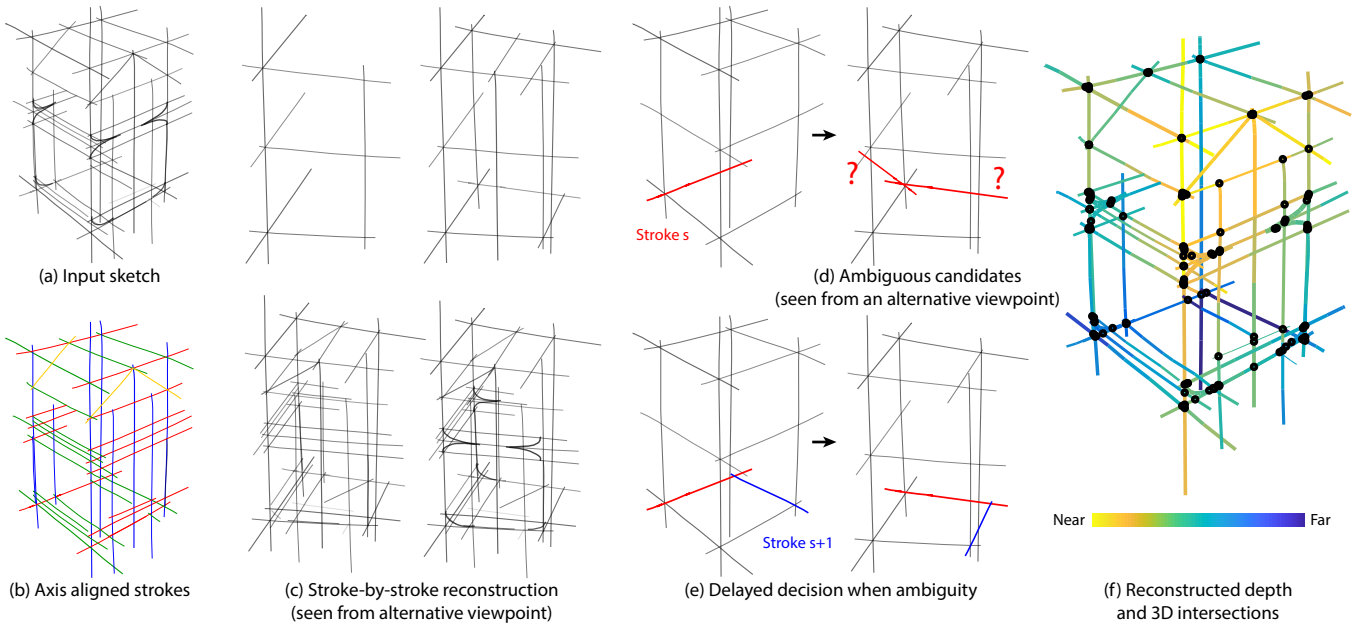


Fig. 3. **Overview of our method.** Given an input sketch (a), we first identify all straight strokes that are parallel to one of the major axes in the drawing (corresponding to its vanishing points) (b). We next reconstruct the 3D scaffold, processing one stroke at a time by identifying its 3D intersections with already reconstructed strokes. Once reconstructed, the scaffold provides geometric context to reconstruct curved strokes (c). For each stroke, our algorithm considers multiple candidate reconstructions and selects the one that best satisfies a set of geometric regularities. If no clear best choice exists (d), we delay decision until additional context resolves the ambiguity (e). The final output of our method is a 3D sketch with its connectivity (f).

to calibrate a perspective matrix that maps the word coordinate system to the image one, as detailed in the supplemental materials.

We encode the 3D interpretation of each straight stroke as an explicit line equation, and the 3D interpretation of planar curved strokes using an explicit plane equation. We constrain axis-aligned strokes to be co-linear to the corresponding world axis, and use the perspective matrix to convert between their world and image coordinates. We incorporate per-intersection constraints that force all strokes that intersect at this intersection to have similar depth. Finally, we constrain the depth of one stroke end point to be zero to anchor the reconstruction. The resulting constrained optimization problem can then be solved using different approaches. For example, Kang et al. [2004] grow a maximum spanning tree of axis-aligned strokes starting at the origin, and then reconstruct the non-axis aligned strokes based on their intersections with already-reconstructed strokes.

While these existing methods assume that the 3D connectivity of the stroke network is readily available, we do not have this connectivity at the start of our depth estimation process. Our method computes the stroke depth incrementally, using the drawing order and the inter-stroke connectivity we gradually compute.

3.3 Recovering 3D connectivity

We denote by $\mathcal{I}$ the set of 2D intersections between the input strokes. Our goal is to label each intersection as a true 3D intersection or as an occlusion, resulting in a configuration space $\mathcal{L} = \{0, 1\}^{card(\mathcal{I})}$ where a 0 value represent an occlusion and 1 represents an intersection. For any label configuration $l = (l_i)_{i \in \mathcal{I}} \in \mathcal{L}$, we can lift the strokes to 3D as described above, assuming that there are enough 3D intersections and axis-aligned strokes to make this problem well posed. Our challenge is to select the label configuration that produces the best reconstruction among all of these candidates. We formulate this problem as the maximization of a score function that measures how well the output 3D stroke network satisfies common geometric properties of scaffolds, such as axis-alignment, orthogonality and planarity. Denoting this score function $Score(\mathcal{X})$, and $Lift(\mathcal{S}, l)$ the 3D reconstruction operator, we obtain the optimization problem

$$\underset{l}{\text{argmax}} \; Score\Big(Lift(\mathcal{S}, l)\Big). \tag{1}$$

Solving the above binary labeling problem in its general form is extremely challenging, for several reasons. First, the solution space is very large, with a number of configurations that is exponential in the number of intersections – which varies between ≈ 200 and ≈ 20000 for the sketches we considered. Second, evaluating the quality of a single binary assignment requires solving a complex constrained optimization to recover the 3D geometry of all strokes, making any general binary solver too expensive. We tackle these challenges by leveraging a number of observations about the choices designers make when constructing their drawings.

3.4 Algorithm overview

Following our observation about the scaffold drawing order, we process strokes in their drawing order and rely on past strokes to infer the 3D interpretation of new strokes, whenever possible. We automatically delay the decision and reassess it after additional strokes are drawn when past strokes are insufficient to identify a dominant interpretation.

Our method consists of three key steps, illustrated in Figure 3. We perform preliminary 2D analysis of the raw input sketches to extract robust geometric information necessary to perform our optimization, including stroke classification into curves, axis-aligned, and other straight lines, and rough grouping of overdrawn strokes (Section 4). We then proceed to reconstruct the polyhedral scaffolds one stroke at a time (Section 5). For each stroke, our algorithm considers all possible binary label configurations for the 2D intersections of this stroke with previously reconstructed ones, it discards all inconsistent configurations, and generates candidate 3D lines for the remaining ones. It then selects the candidate that maximizes a score function that measures the degree to which the stroke aligns with the priors listed in Section 3.1. We mark the intersections lying along this line as 3D intersections, and others as occlusions. We extend our algorithm to handle the common case where several interpretations yield a similar high score (Section 5.4), for which we propose a dedicated search algorithm that progresses over the drawing sequence until one of these concurrent interpretations emerges as demonstrably better than the others. Once the scaffold is recovered, we reconstruct planar curved strokes following the same logic (Section 6); we generate candidate planar 3D curves for each stroke based on its intersections with the scaffold and already reconstructed curves, and select the candidate that yields the shape most aligned with our priors.

4 2D ANALYSIS

We perform basic 2D analysis on our input data to facilitate subsequent 3D estimation. We first separate straight strokes from curved strokes since we reconstruct the former as a preliminary step for reconstructing the latter. We further classify the straight strokes into axis-aligned and non axis-aligned by detecting the three bundles of strokes converging towards vanishing points [Hedau et al. 2009]. These vanishing points allow us to calibrate a perspective camera [Guillou et al. 2000; Orghidan et al. 2012], which gives us the 3D direction of the associated orthogonal axes. Finally, we also identify ellipses among curved strokes, which we assume to correspond to 3D circles.

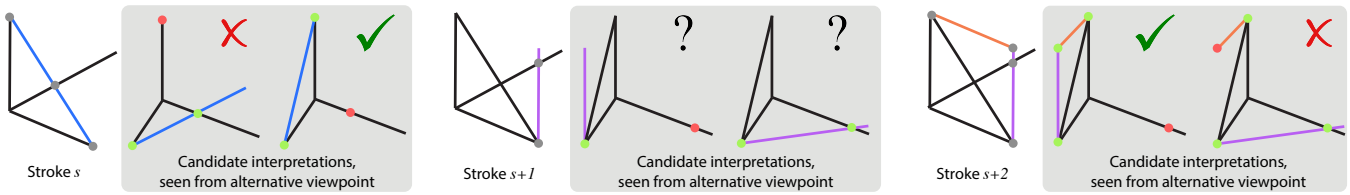
The sketches we target contain many overdrawn strokes. We consolidate this raw data to reduce the complexity of the subsequent 3D reconstruction and generate longer strokes more likely to correspond to intended geometric curves. Specifically, we define for each stroke a neighborhood size proportional to its velocity, based on the observation that fast strokes are less precise. We merge strokes if one is within the other's neighborhood by more than 75% of its length and if they form an angle of less than 5°. We restrict straight-stroke merging to strokes that are drawn in sequence, *i.e.*, less than five strokes apart, to prevent merging accidentally partly-overlapping strokes that correspond to different parts of the scaffold. For curved strokes, we perform a more aggressive clustering using the conservative version of the *StrokeAggregator* [Liu et al. 2018] method. During processing we use the time-stamp of the first drawn stroke in a cluster as the time-stamp of that cluster. Note however that while we use consolidated strokes for internal computation, we re-project the recovered depth on the original strokes for rendering to preserve the input sketch style. See Section 7.4 for a study of the impact of stroke consolidation on our results.

We compute a consolidated set of 2D intersections, overcoming noise in the drawings. We classify almost-intersecting strokes as intersecting if the end point of one stroke is within the neighborhood of another or vice versa. Due to the inaccuracy of sketching, multiple coincident lines often do not intersect exactly at the same point, but rather result in a multitude of nearby intersections. We recover the local 2D connectivity of the stroke network by associating with each intersection all strokes incident to nearby intersections. We consider that two intersections are close enough if they lie along the same stroke and if each intersection is within the neighborhood of the other, where we define the neighborhood size of an intersection as the neighborhood size of the last-drawn stroke incident to that intersection.

Finally, we perform trivial intersection filtering, classifying 2D intersections that are highly unlikely to correspond to real 3D intersections as occlusions. This step reduces our set of unknowns, speeding up computation and improving performance (Figure 8). We observe that, due to the dense 3D connectivity of scaffolds, accidental occlusions most often occur between two isolated strokes, while true 3D intersections are formed by three or more strokes. The notable exceptions are intersections near stroke end-points, and tangential intersections between curves or between curves and straight lines. Following this observation, we mark intersections formed by three or more differently-oriented strokes, or tangential intersections that include curved strokes as *unclassified*, and mark the rest as *occlusions*. We extend the set of unclassified intersections to include pairwise intersections near stroke end points if the nearest potential intersection along the stroke is more than 1/4 of the stroke's length away. This extension ensures that our set of unclassified intersections includes at least two intersections along every stroke that has at least two intersections in image space. These heuristics allow us to reduce the problem dimensionality from several thousands to a few hundred unknowns, on average. Please refer to supplemental materials for implementation details of these steps.

5 RECONSTRUCTING STRAIGHT STROKES

Our reconstruction processes one stroke at a time using the drawing order. For every new stroke, we first generate all of its possible interpretations and then select the best one using a score function motivated by the observations in Section 3.1. When multiple interpretations garner similarly high scores, we retain all such interpretations and move on to the next stroke, until one of the interpretations becomes clearly better than the others, keeping track of multiple potential solutions as we go. We now detail each of these components. Figure 4 provides a schematic illustration of our algorithm.



(a) Unambiguous stroke (b) Ambiguous stroke (c) Joint assignement

Fig. 4. **Intuition behind our approach.** (a) The new stroke s, shown in blue, intersects several already-reconstructed strokes, shown in black. Each pair of intersection gives a candidate reconstruction of the stroke, shown from an alternative viewpoint. One interpretation lies in the ground plane and extends way beyond its second intersection. The second interpretation lies in a vertical plane and is well bounded by its intersections. We select the second interpretation since as we observed, designers rarely draw strokes much longer than their intended length. The middle intersection is thus identified as being an occlusion. (b) The next stroke $s+1$, shown in purple, has two good candidates. One has a high score because it is aligned with a vertical axis, while the other one has a high score because it is well bounded by two intersections. Given this ambiguity, we leave both options open. (c) Stroke $s+2$, shown in orange, has a single candidate aligned with an horizontal axis. This new stroke helps disambiguate the previous one by providing an additional intersection that makes the vertical interpretation well bounded.

5.1 Candidate Lines

Let us first consider the general case where we have successfully reconstructed all strokes until the current one. Let the new stroke have N unclassified 2D intersections with the reconstructed strokes. Note that the geometry of an axis-aligned line is determined by one intersection and the geometry of other lines by two. We first generate for all lines $\frac{N(N-1)}{2}$ candidate 3D lines passing through all possible pairs of intersections. For each axis-aligned line we augment the set of candidates by creating N additional lines, each going through one of the intersections and parallel to the associated coordinate axis. We augment rather than replace the candidate set so that it also contains non-axis aligned candidates, which provides robustness to inexact axis classification and perspective inaccuracy. For each candidate line we detect all other 2D intersections along it where the 3D position of the intersection is within ϵ-distance from a reconstructed intersecting stroke, and include these intersections in the candidate's 3D intersection set. We fix ϵ to 10% of the length of the line. In practice, the above procedure produces many similar candidates when multiple pairs of intersections align in 3D. We merge such redundant lines whose 3D intersection sets coincide, averaging their end points.

Before proceeding with candidate selection, we perform preliminary clipping based on foreshortening, rejecting candidates for which the 3D length h_{3D} is too large compared to the length of the 2D stroke h_{2D}, except if no other candidates exist. Since different sketches might exhibit different amounts of foreshortening, we use the already reconstructed strokes as a prior on the depth distribution of the scene, as also proposed by Schmidt et al. [2009b]. Specifically, we compute the median 2D and 3D length of the reconstructed strokes, $\overline{h}_{2D}$ and $\overline{h}_{3D}$, and reject candidates for which the deviation from median in 3D is much higher than in 2D, as measured by $\frac{h_{3D}}{\overline{h}_{3D}} > 5\frac{h_{2D}}{\overline{h}_{2D}}$.

5.2 Candidate Evaluation

We evaluate each candidate 3D line using a combination of criteria motivated by our observations of how designers draw scaffolds (Section 3.1).

Coverage. Following the observation that designers avoid overshooting lines past their intended end points, we prioritize candidate lines whose intersection sets contain intersections that are as far away as possible from one another. Our coverage score consequently measures the portion of the processed stroke covered by the segment delimited by the most distant 3D intersections along the candidate line:

$$Q_{coverage} = \frac{d_{candidate}}{d_{input}}, \tag{2}$$

where $d_{candidate}$ is the distance between the projection of the two most distant 3D intersections, and d_{input} the distance between the two most distant intersections along the input stroke. This definition of d_{input} trims dangling segments past the first and last 2D intersections because we consider those segments not to be intentional.

Axis alignment. For strokes classified as axis-aligned, the key criterion for assessing the candidate 3D line is the degree to which it coincides with the intended axis. Denoting $\mathbf{l}$ the direction of the 3D line and $\mathbf{a}$ the direction of the axis, we define the score function as

$$Q_{axis}(\mathbf{l}, \mathbf{a}) = G(1 - |\mathbf{l} \cdot \mathbf{a}|, \sigma_1), \tag{3}$$

where $G(\cdot, \cdot)$ denotes the Gaussian function and σ controls the tolerance to misalignment. As we seek for this score to drop to 0 when the angle between the axis and the stroke exceeds $\alpha = 15°$, we set $\sigma_1 = (1 - cos(\alpha))/3$ using the three sigma rule.

Non-axis-aligned strokes. For strokes not expected to be aligned with major axes, we prioritize candidate 3D interpretations that satisfy one of the following three common regularities provided by scaffold embedding.

Orthogonality. For each 3D intersection i along a candidate line, we favor orthogonality with any adjacent line using the following score function:

$$Q_{ortho}(\mathbf{l}, i) = \max_{k \in \mathcal{N}(i)} G(|\mathbf{l} \cdot \mathbf{l}_k|, \sigma_2), \tag{4}$$

where $\mathcal{N}(i)$ denotes the set of 3D lines incident to intersection i. We want this score to drop close to 0 when the angle between the

directions deviates from being orthogonal by more than α, thus we set $\sigma_2 = cos(90 - \alpha)/3$ following the three sigma rule.

We compute the total orthogonality score of a line as the maximum over all of its 3D intersections:

$$Q_{ortho}(\mathbf{l}) = \max_i Q_{ortho}(\mathbf{l}, i). \tag{5}$$

Planarity. We favor local planarity of the polyhedral scaffold by encouraging the candidate line to lie in one of the planes defined by pairs of adjacent lines. We measure this regularity criterion at each 3D intersection along the line using the score function

$$Q_{planar}(\mathbf{l}, i) = \max_{(m,n)\in\mathcal{N}(i)} G(|\mathbf{l} \cdot (\mathbf{l}_m \times \mathbf{l}_n)|, \sigma_2), \tag{6}$$

where (m, n) is any pair of incident lines to intersection i. We compute the total planarity score as a maximum over all intersections along the line.

Tangentiality. To account for the case when one continuous line was drawn as several strokes that were not merged together during the 2D analysis step, we evaluate tangentiality of two strokes:

$$Q_{tangent}(\mathbf{l}, i) = \max_{k\in\mathcal{N}(i)} G(1 - |\mathbf{l} \cdot \mathbf{l}_k|, \sigma_2). \tag{7}$$

We aggregate axis alignment, orthogonality, planarity and tangentiality scores into a single *geometric* score

$$Q_{geom}(\mathbf{l}) = \begin{cases} Q_{axis}(\mathbf{l}, \mathbf{a}) & \text{when } \exists\ \mathbf{a} \\ \max(Q_{ortho}(\mathbf{l}), Q_{planar}(\mathbf{l}), Q_{tangent}(\mathbf{l})) & \text{otherwise.} \end{cases} \tag{8}$$

Total score. We define the total score of a candidate line as:

$$Q = 0.4Q_{geom} + 0.4Q_{coverage} + 0.2Q_{geom}Q_{coverage}, \tag{9}$$

where the last term favors lines that simultaneously satisfy both coverage and geometric scores over the lines that maximize only one of the two score functions.

Acceptance criteria. We define the best candidate line as the one that maximizes the total score. However, not all strokes can be reliably lifted into 3D using previously drawn strokes alone. We determine whether we should accept the best candidate or delay decision by accounting for two complementary criteria: we want the accepted candidate line to have a high score, *i.e.*, to be well aligned with our geometric and topological priors; and we also want this candidate line to stand out compared to alternatives. We balance these two considerations by accepting the best candidate lines with exceptionally high scores, and accepting others if their scores are both sufficiently high and are significantly higher than those of all other candidates. Specifically, we accept the best candidate reconstruction if its score is above an acceptance threshold $\tau_{score} = 0.75$, and if the ratio between the score of the second best candidate and the best score is below an ambiguity threshold $\tau_{ambiguity} = 0.8$. In addition, we also accept all best candidates if their score exceeds $\tau_{scoreHigh} = 0.98$. In summary, our acceptance criterion is

$$Accept(Q^{best}, Q^{2^{nd}best}) := (Q^{best} > \tau_{scoreHigh}) \vee \left((Q^{best} > \tau_{score}) \wedge (\frac{Q^{2^{nd}best}}{Q^{best}} < \tau_{ambiguity})\right). \tag{10}$$

Section 5.4 discusses the handling of strokes whose best candidate line does not satisfy the acceptance criteria.

5.3 Initialization

We can always fix the depth of one intersection in our drawing arbitrarily. However, while an axis-aligned line requires one depth value to anchor it in space, a non-axis aligned one requires two. To obtain the first stroke that we can reliably reconstruct, we search for the first intersection between strokes aligned with different axes in the drawing sequence and use it to anchor the drawing in space. We assume this intersection to lie on the ground plane, which fixes its depth relative to the camera and defines the positions of the participating orthogonal strokes. We then process the preceding strokes in the order in which they were drawn, treating them as if they had been drawn after the anchoring strokes.

5.4 Processing Ambiguous Strokes

If a stroke's best candidate line does not satisfy the acceptance criteria, we delay the assignment decision. Our algorithm then explores several solutions in parallel, each depending on a different candidate interpretation of the ambiguous stroke.

Stroke dependency graph. Each candidate line of an ambiguous stroke results in different candidate interpretations for the next strokes. We keep track of these different solutions by constructing the *dependency graph* of ambiguous strokes, where two strokes are considered to depend on each other if they share a 2D intersection, as illustrated in Figure 5. Each solution corresponds to a different combination of candidates, and as such to a different *joint score* of the ambiguous set, $\overline{Q}$, defined as the average score of the corresponding 3D lines. Note in particular that the candidates of each new stroke impact the scores of the candidates on which they depend, since they add new intersections along them. Given the combinatorial nature of our problem, we devised several strategies to contract this dependency graph and maintain the scalability of our method.

Rejection of inconsistent combinations. While the space of combinations of candidate lines is vast, many are not consistent. In the example shown in Figure 5(c), while the 6^{th} stroke depends on the 2^{nd} and 4^{th} strokes, these two strokes both depend on the 1^{st} stroke. To be consistent, the candidates of the 6^{th} stroke need to form combinations with candidates of the 2^{nd} and 4^{th} strokes that themselves are derived from the same candidate of the 1^{st} stroke (depicted as dashed or solid lines). We only generate such candidates.

Grouping of redundant candidates. A stroke can have multiple candidates that form a similar line in 3D, yet intersect different candidates of other ambiguous strokes. Due to their similar geometry, these candidates share the same 3D intersections with already-assigned strokes. Since our orthogonality, planarity and tangentiality scores (Eq. 4, 6, 7) are computed at each 3D intersection along a candidate line, we only compute these scores once for shared intersections. We consider that two candidates form similar lines in 3D if their endpoints are at 10% of the shortest-length distance of each other. Similarly, when updating the joint score $\overline{Q}$ for every new candidate, we only re-compute the scores on intersections impacted by that candidate.

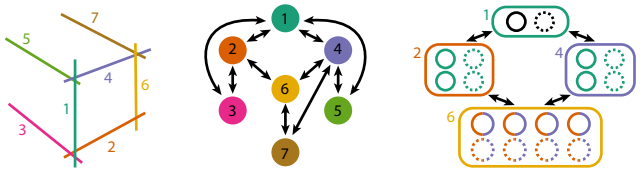


Fig. 5. **Schematic illustration of the stroke dependency graph**. Each stroke points to the strokes on which it depends (a,b). For illustration purposes, we assume that each new stroke generates two 3D candidate lines for each version of the preceding strokes on which it depends (c). We depict the multiple candidates of each stroke with a disk, and represent the dependency of a candidate to preceding ambiguous strokes by color-coding the disk according to its ancestors (c). Each such candidate results in a different joint score $\overline{Q}$ of all the strokes on which it depends. When an unambiguous configuration emerges, we assign all strokes that contribute to it.

Early acceptance. We attempt to assign ambiguous strokes every time a new stroke is added to the sketch, by testing if a subset of strokes containing this stroke can be classified as no longer ambiguous. We consider that a set of strokes is unambiguous if both the score of the best candidate of the last added stroke and the joint score of its dependencies satisfy the acceptance criteria

$$\left(Q^{best} > \tau_{scoreHigh}\right) \vee Accept(\overline{Q}^{best}, \overline{Q}^{2^{nd}best}). \quad (11)$$

If these criteria are satisfied, we assign to each stroke the candidate line that corresponds to the best score and prune the dependency graph accordingly. If this is not the case, we proceed with the next stroke and insert it in the dependency graph.

Candidate set trimming. As an additional precaution against computational explosion, we attempt to assign ambiguous strokes using a more permissive criteria if the expected number of candidate interpretations of the new stroke exceeds a threshold, fixed to 400. When this situation occurs, we revisit all ambiguous strokes on which the new stroke depends and greedily assign the strokes that have high-score candidates until the number of candidates of the new stroke falls below the threshold.

Finalization. We resort to greedy assignment for strokes that remain ambiguous at the end of the straight stroke sequence.

6 RECONSTRUCTING CURVED STROKES

The last part of our method reconstructs the curved strokes based on their intersections with the scaffold lines and already-reconstructed curves. Similar to straight strokes, we process the curved strokes one by one in their order of appearance. However, as with straight strokes, we again face the challenge of identifying which intersections truly occur in 3D, and which are only due to occlusions. We identify the true 3D intersections by applying the same solution strategy as for straight lines, *i.e.*, we compute multiple per-stroke candidate 3D reconstructions and select the one which yields the 3D curve most consistent with our observations about scaffold drawing practices. In the text below we only address the differences between the straight and curved stroke handling. All other computations are identical.

6.1 Candidate Planes

We leverage the observation that design sketches mostly feature planar curves embedded into scaffold planes, which allows us to shift the problem of generating general 3D candidate curves to the easier task of generating *candidate planes*, onto which we project the input strokes.

As a pre-process, we compute all scaffold planes formed by any pair of straight strokes incident to a scaffold 3D intersection, and we associate each stroke with all the planes it lies in. We also compute the bounding box of all scaffold 3D intersections, which provides a bounding structure for the reconstructed curves.

For each curved stroke, we then consider all 2D intersections with straight strokes and retrieve their associated scaffold planes. We generate additional candidates by projecting each curve to the three axis-aligned planes passing through each intersection. Finally, we merge nearby planes to avoid redundant candidate curves. We consider that two planes can be merged if they contain the same 3D intersections and have orientations that differ by less than 20°, where we consider that a plane contains an intersection if this intersection is at a distance of less than 2% of the bounding box diagonal. After evaluating the candidate curves with respect to our score function, we select the highest scoring one and mark all intersections between the curve and previously reconstructed strokes which lie in the corresponding plane as 3D intersections.

Similarly to straight strokes, we reject obvious bad candidates before proceeding with candidate selection. We exploit the fact that scaffolds often act as bounding volumes to reject candidates that lie outside of the bounding box of scaffold intersections, dilated by 10% for safety. Yet, some sketches do contain curves outside the scaffold. We detect these cases by 2D analysis, comparing the convex hull $\mathcal{H}_s$ of the projected scaffold intersections with the convex hull of the curved strokes $\mathcal{H}_c$. When the ratio $\frac{\mathcal{H}_c}{\mathcal{H}_s \cup \mathcal{H}_c}$ is greater than 1, we consider a bounding volume twice the size of the bounding box to allow for curves outside the scaffold.

6.2 Score function and acceptance criteria

Since curves are typically drawn after straight lines, they often suffer from higher clutter. In addition, since curves are geometrically more complex than lines, designers often utilize multiple scaffold-curve intersections to ease accurate curve drawing and enable viewers to accurately perceive them. Thus, curved strokes often have multiple 3D intersections with the scaffold, each providing complementary regularity cues about the intended shape. At the same time, due to stroke clutter buildup, curved strokes have more accidental intersections. We adapt our score function to these characteristics.

Orthogonality and tangentiality. Our geometric term accounts for the higher number of intersections along curved strokes. While for straight strokes we seek to have at least one intersection with high orthogonality, tangentiality or planarity along a candidate line (Equation 8), for curved strokes we seek to maximize the total number of tangential or orthogonal intersections, as measured by

$$Q_{curve\ geom} = \frac{1}{N} \sum_i \delta_i, \quad (12)$$

with $\delta_i = 1$ if the i^{th} intersection is tangential or orthogonal, $\delta_i = 0$ otherwise. We consider that an intersection is tangential or orthogonal if the angle formed by the two tangents is below 30° or above 80°, respectively.

Foreshortening. We complement the geometric term with a term that encourages interpretations that face the camera, motivated by the *minimal foreshortening* principle of design sketching:

$$Q_{foreshortening} = |\mathbf{p} \cdot \mathbf{v}|, \tag{13}$$

where $\mathbf{p}$ denotes the normal of the candidate plane and $\mathbf{v}$ the camera direction.

Coverage. The coverage term computes the ratio between the length of the 2D stroke segment bounded by the most far apart 3D intersections, and the length of the input 2D stroke.

In contrast to straight lines, for which we seek to satisfy coverage and geometry equally, we give a higher weight to coverage for curved strokes because it is a more reliable cue in the presence of multiple intersections. We define the total score as,

$$\begin{aligned} Q =& 0.6 Q_{coverage} + 0.2 Q_{curve\ geom} \\ &+ 0.2 Q_{foreshortening}. \end{aligned} \tag{14}$$

Ellipses. We employ a slightly different score function for ellipses, which are frequently used to represent cylindrical parts. We favor the most circular ellipse by encouraging the radii of all the points along the ellipse to be close to the maximum radius of that ellipse:

$$Q_{circularity} = \frac{1}{P} \sum_i \frac{r_i}{r_{max}}$$

where P is the total number of points and r_i is the radius of the i-th point. We use this term in place of the coverage term, which is not well defined for closed curves, yielding

$$Q = 0.8 Q_{circularity} + 0.2 Q_{curve\ geom}. \tag{15}$$

Non-planar curves. Non-planar curves over man-made objects can often be broken into planar segments. We have implemented a simple step to detect potential non-planar curves in our aggregated strokes, which we break into individual strokes. To do so, we first collect all tangential intersections along the first and last quarter of the curve, which represent candidate intersections for the curve extremities. We then test the planarity of all pairs of such extremities. If no candidate pair of tangents form a plane, we classify the aggregated curved stroke as planar and break it.

7 EVALUATION AND RESULTS

We first compare the performance of our intersection labeling against human annotations, as a mean to evaluate the perceptual relevance of our algorithm. We also evaluate accuracy by annotating erroneous strokes in all of our results. We then evaluate the impact of different components of our method, before presenting results on a variety of real-world industrial design sketches, and potential applications for design exploration.

7.1 Comparison to human labels

While the sketches we used to test our method were drawn according to reference 3D models [Gryaditskaya et al. 2019], they do not come with ground truth depth. Not only are the sketches not perfectly aligned with the reference due to drawing inaccuracy, the scaffold lines we are interested in simply do not lie over the reference surface. As an alternative, we quantify the plausibility of our solutions via a perceptual study, where we asked 13 participants to classify intersections in sketches as occlusions or 3D intersections. All participants annotated 3 sketches of varying complexity, shown in Figure 6. Each sketch was displayed one stroke at a time in their original order, and participants were asked for each newly added stroke to label its intersections before moving to the next stroke. We also allowed participants to see the complete sketch at any time to give them global context. To reduce fatigue, we only had participants label a subset of intersections per sketch (150 for the house, 110 for the shampoo bottle, and 50 for the bump surface).

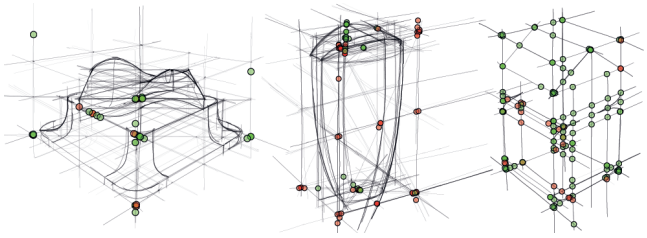


Fig. 6. We asked participants to label a subset of intersections in three sketches of varying complexity. Green disks correspond to intersections on which our algorithm agrees with the majority vote, red disks are intersections on which our algorithm disagrees. The disks are rendered with transparency to depict overlay between nearby intersections (best seen in digital version).

We first compute the majority labeling by taking for each intersection the decision on which more than 50% of participants agree. On average, participants show an agreement of 89.1% with the majority labeling, showing that the task is difficult even for humans. On average, our algorithm agrees by 78.1% with human annotations. Moreover, our algorithm is more consistent with the majority labeling for intersections on which participants reached a higher agreement. For intersections with human agreement between 50% and 75%, our method is consistent with human annotations just by 48.4%, while for intersections with human agreement above 75%, our method reaches an agreement of 86.2% with human annotations. In other words, the cases that are most difficult for our algorithm are also difficult for humans.

While our participants were not professionals, we also asked a designer to perform the task. The agreement between the designer labels and ours is 78.7%, and the agreement between the designer labels and the majority label of other participants is 89.5%, marginally higher than the agreements obtained by non-professionals. This small difference is consistent with similar studies [Shao et al. 2011] that reported that non-experts perform on par with designers when interpreting drawings (they found a 2 degree difference in accuracy between lay people and designers in interpreting surface orientation).

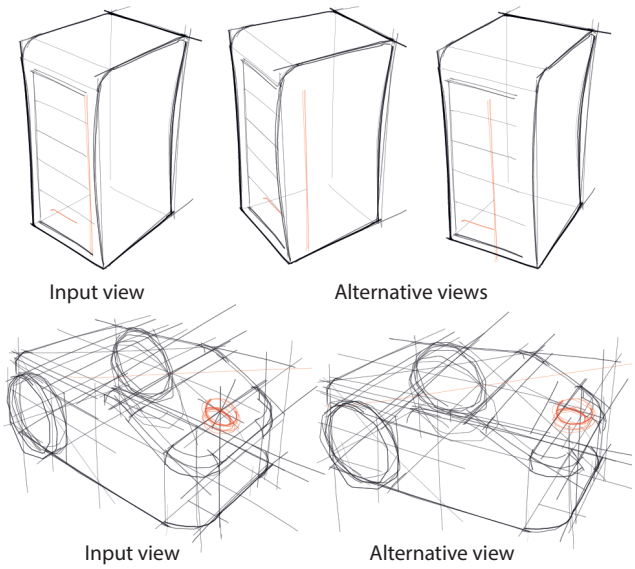


Fig. 7. We asked 3 participants to annotate spurious erroneous strokes across all our results (Section 7.2), shown in red. Line width has been adjusted for improved visibility.

7.2 Accuracy

To further quantify the accuracy of our method, we asked 3 participants to annotate spurious erroneous strokes across all of our results using a 3D interface and using the input drawings as reference. Figure 7 illustrates typical errors identified by the participants. The median percentage of such erroneous strokes was only 7.95%. We observed that 31% of erroneous scaffold strokes were among those whose processing was delayed until the end due to low score and/or confidence (median percentage), against 19% for good strokes. This means that these spurious errors had limited impact on other strokes. We discuss the potential sources of such errors in Section 7.5.

7.3 Computation time.

Our research prototype is implemented in Matlab and Python, and has room for optimization. The computation time for scaffold reconstruction varies between half a minute to 1.5 hours for the sketches we considered, with a median time of ≈ 3 minutes. Timings correlate strongly with both the number of intersections and the number of strokes processed. The Spearman correlation coefficient between computation time and number of strokes is 0.87, and between computation time and number of intersections is 0.89, with p-value less than $1e-26$ in both cases. The runtime for the curved strokes reconstruction varies between ≈ 2 and ≈ 85 minutes, with a median time of 12 minutes. The total number of strokes varies between a few dozens to a few hundreds after consolidation, while the number of intersections varies between ≈ 40 and ≈ 4000 after filtering.

Given this median reconstruction time of a dozen of minutes, a variant of our method could potentially run as a background task during sketching, where camera parameters could be estimated a soon as a sufficient number of mutually orthogonal lines are present, and where offline stroke consolidation could be replaced by an interactive stroke clustering algorithm.

7.4 Ablation studies

We compare our method to several variants by disabling some of its components. For simplicity, we only processed straight strokes when performing these experiments.

Impact of intersection filtering. As mentioned in Section 4, we perform trivial intersection classification to reject obvious occlusions prior to applying our optimization. This filtering reduces the average computation time of our method from 13 minutes to 8.5 minutes per sketch compared to solving for all intersections. On average, the solutions obtained with this filtering agree by 90% with the solutions obtained without, showing the robustness of our method to accidental occlusions. Nevertheless, we observed that intersection filtering improves our results overall, as illustrated in Figure 8.

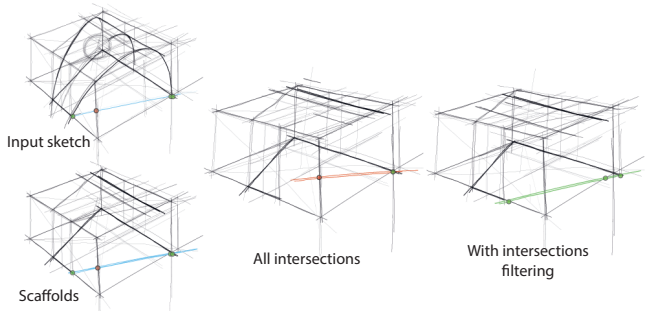


Fig. 8. Rejecting intersections between less than 3 strokes prior to optimization not only accelerates the algorithm, it also improves its performance, since most such intersections correspond to occlusions.

Impact of delaying decision. An important contribution of our work is the careful treatment of ambiguity by delaying decision as we progress along the sketching sequence. In Figure 9 we plot the distribution of delayed decisions on the full dataset – most ambiguous strokes are resolved after just a few extra strokes, with a median delay of 7 strokes. Nevertheless, 23.5% of strokes remain ambiguous at the end of the sequence, and are assigned greedily. Figure 10 compares our approach to a greedy algorithm that selects the best candidate of each stroke as soon as it is drawn, akin to the algorithm by Schmidt et al. [2009b]. The greedy algorithm interprets many of the hidden lines has being part of the front of the object. The intersection labels found by the greedy algorithm agree by 71.6% with the ones obtained by our method, while the two algorithms only differ by a few seconds in terms of running time, on average.

Impact of candidate set trimming. To prevent computational explosion, our algorithm switches to greedy assignment of ambiguous strokes when the number of candidates of the new stroke to process exceeds a threshold. Disabling this safeguard increases computation time by a factor of 2.6 on average, although the effect is more dramatic on sketches that exhibit more ambiguity, some of which can take up to several hours to process. The solutions obtained with and without trimming agree by 85%, on average.

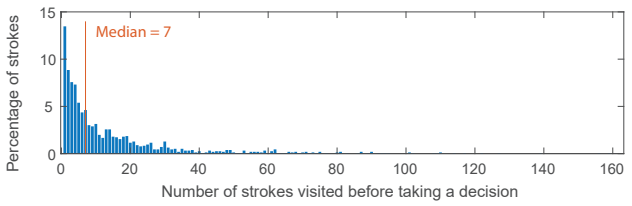


Fig. 9. Histogram of delayed decisions. 57% of the strokes get assigned as soon as they are drawn. Among the remaining ambiguous strokes, around 14% are assigned after considering one additional stroke. The median delay of ambiguous strokes is 7 strokes.

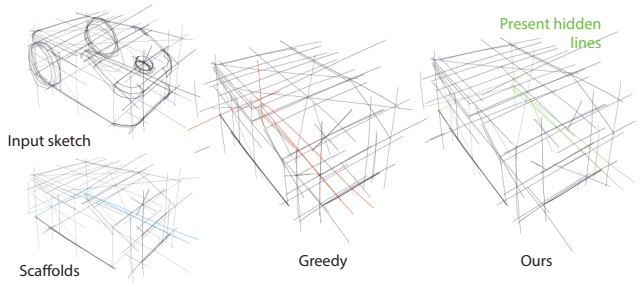


Fig. 10. Comparison between our algorithm that delays decision in the presence of ambiguity, and a greedy algorithm that immediately selects the best candidate for each stroke. The greedy algorithm often misinterprets occlusions as intersections, bringing hidden lines to the front of the shape.

Impact of stroke consolidation. While conservative, the stroke consolidation we perform prior to optimization accelerates the reconstruction of straight strokes by a factor of 3, on average, yet has little impact on quality. Aggregation is more critical for curve reconstruction as it allows multiple strokes to share a common planar interpretation, as illustrated in Figure 11.

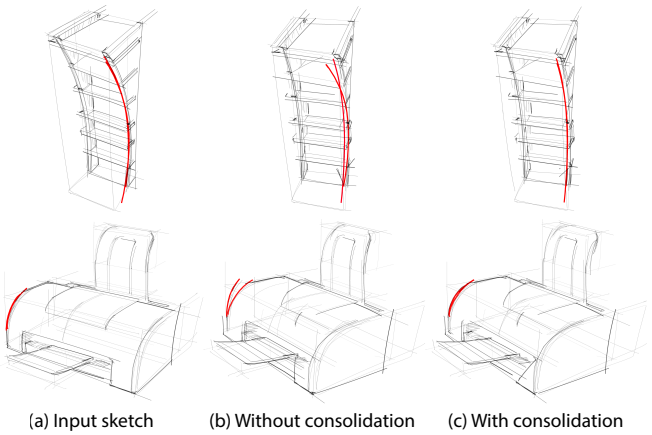


Fig. 11. Our algorithm benefits from preliminary stroke consolidation, as multiple strokes form longer curves that share geometric constraints.

7.5 Results and limitations

Figure 15 presents a gallery of sketches processed by our method, which form a subset of the 47 sketches that we provide as supplemental materials. A large number of these sketches are from the *OpenSketch* dataset [Gryaditskaya et al. 2019], which contains 12 shapes, each drawn by several professional designers. In addition, we collected 12 sketches of complex man-made objects.

We visualize the reconstructed 3D information in the form of a depth map, as well as via alternative viewpoints. We also provide turntable animations of all our results as supplemental materials. While these animations showcase the rich 3D structure recovered by our method, spurious erroneous strokes become visible when moving far away from the original viewpoint. Figure 7 and 12 illustrate typical errors.

We identified several sources of incorrect or distorted interpretations. One source of error is ambiguity between multiple solutions, such as on the side of the car (Figure 12, bottom). Another source of error is when a stroke lacks intersections with other strokes to be well-positioned in 3D, such as the bowl of the mixer (Figure 12, top) or the front of the car (Figure 12, bottom). Finally, a third source of error is stroke consolidation, which sometimes merges strokes that should not. A possible direction for future work would be to exploit symmetries, for both scaffold and curves. However, designers often do not draw symmetric objects accurately [Schmidt et al. 2009a], and we have observed that they tend to draw more details on the front-facing half of the object, which makes the detection of symmetric correspondences a challenge.

Our method imbues strokes with depth while keeping their 2D locations fixed. While this is necessary for the image-processing applications we target, this choice bakes-in any and all artist errors and inaccuracies. Ignoring such inaccuracies allowed us to focus on the under-explored problem of automatically inferring the 3D connectivity of the drawing, without considering the separate problem of 2D correction. Prior work [Xu et al. 2014] has shown that quality 3D reconstruction requires modifying the 2D locations of the reconstructed strokes to correct for sketching inaccuracy; we believe that their selective regularization could be applied to our results in future work.

7.6 Applications

The sparse 3D information we extract from raw sketches opens the door for a number of novel 3D-aware sketch editing functionalities. Inspired by popular single-image photo pop-up applications [Hoiem et al. 2005; Niklaus et al. 2019], we provide as supplemental materials animations of all our results under small camera changes, which provide a vivid sense of the 3D shapes in the sketches while reducing visual artifacts.

Figure 13 demonstrates a shape editing application, where we scaled the 3D sketch along one of its major axis. Note how our results preserve coherent perspective, occlusions and parallax. In contrast, Photoshop perspective-aware warp [Adobe 2020] fails to produce a convincing edit. See supplemental materials for animations of scale editing on all our results.

Figure 14 demonstrates the potential of our method to support the industrial design workflow. Thanks to our 3D reconstruction, the

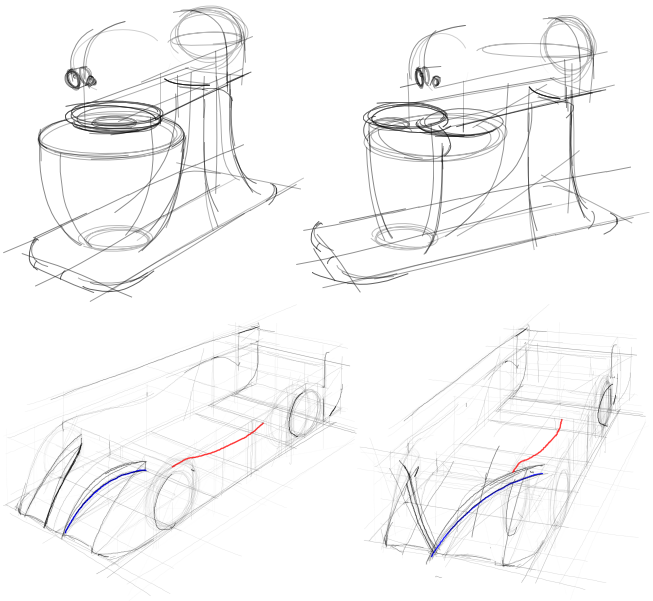


Fig. 12. **Limitations.** When the sketch only contains very few scaffold lines, our method misplaces curves, as for the bowl of this mixer (top). Even when the scaffold is dense, some curve may lack support, as for the front part of the car (bottom, blue). Some curves may also be misplaced because they intersect several scaffold planes, as for the side curve of the car that has multiple 2D intersections with the ground plane of the scaffold (bottom, red).

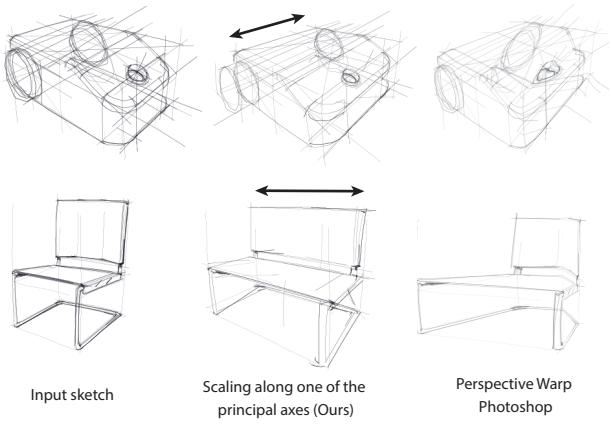


Fig. 13. Our method allows to scale the drawn shape along its principal 3D axes, while preserving perspective, occlusion and parallax. Compared to out method, the Perspective Warp from Adobe Photoshop unrealistically distorts the shape.

designer only needs to sketch a concept once to visualize it under novel viewpoints and to experiment with different proportions. The new sketches generated by our method can form the basis of novel design iterations, or can be used as underlays to create shaded presentation drawings for communication with team members and clients.

8 CONCLUSION

We have presented the first algorithm capable of lifting real-world industrial design sketches to 3D automatically. We achieved this goal by leveraging a number of observations on how designers construct their drawings – in particular using straight construction lines called *scaffold* – and by accounting for ambiguity at multiple stages of our method, from initial 2D stroke processing to core 3D inference. We have applied our method on a large number of sketches captured under an uncontrolled setting, and have demonstrated potential usage of the recovered 3D information for several applications in design exploration and communication. We hope that this research will stimulate the emergence of other 3D-aware sketching tools, just as automatic computer vision has revolutionized modern photography.

ACKNOWLEDGMENTS

This work was supported by the ERC Starting Grant D3 (ERC-2016-STG 714221), NSERC and by software and research donations from Adobe. The authors are grateful to Inria Sophia Antipolis - Méditerranée "Nef" computation cluster for providing resources and support (https://wiki.inria.fr/ClustersSophia). We thank Bastien Wailly for his help on intermediate data visualizations and Valentin Deschaintre for the final video compilation; Enrique Rosales, Yuan Yao and Jerry Yin for data annotation, as well as all the anonymous participants of the perceptual study and the designers who created concept sketches for our work.

REFERENCES

Adobe. 2020. Photoshop Perspective Warp. https://helpx.adobe.com/photoshop/using/perspective-warp.html.

Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. 2008. ILoveSketch: as-natural-as-possible sketching system for creating 3d curve models. In *Proc. UIST*. ACM, 151–160.

Mikhail Bessmeltsev and Justin Solomon. 2018. Vectorization of Line Drawings via PolyVector Fields. *arXiv preprint arXiv:1801.01922* (2018).

Alexandra Bonnici, Alican Akman, Gabriel Calleja, Kenneth P Camilleri, Patrick Fehling, Alfredo Ferreira, Florian Hermuth, Johann Habakuk Israel, Tom Landwehr, Juncheng Liu, et al. 2019. Sketch-based interaction and modeling: where do we stand? *Artificial intelligence for engineering design analysis and manufacturing* (2019), 1–19.

Tao Chen, Zhe Zhu, Ariel Shamir, Shi-Min Hu, and Daniel Cohen-Or. 2013. 3-Sweep: Extracting Editable Objects from a Single Photo. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 32, 6 (2013).

Xuejin Chen, Sing Bing Kang, Ying-Qing Xu, Julie Dorsey, and Heung-Yeung Shum. 2008. Sketching reality: Realistic interpretation of architectural designs. *Trans. on Graphics* 27 (2008). Issue 2.

Frederic Cordier, Hyewon Seo, Mahmoud Melkemi, and Nickolas S. Sapidis. 2013. Inferring Mirror Symmetric 3D Shapes from Sketches. *Computer Aided Design* 45, 2 (Feb. 2013), 301–311.

Johanna Delanoy, Mathieu Aubry, Phillip Isola, Alexei Efros, and Adrien Bousseau. 2018. 3D Sketching using Multi-View Deep Volumetric Prediction. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1, 21 (2018).

Julie Dorsey, Songhua Xu, Gabe Smedresman, Holly Rushmeier, and Leonard McMillan. 2007. The Mental Canvas: A Tool for Conceptual Architectural Design and Analysis. In *Proc. IEEE Pacific Conference on Computer Graphics and Applications*. 201–210.

Koos Eissen and Roselien Steur. 2008. *Sketching: Drawing Techniques for Product Designers*. Bis Publishers.

Koos Eissen and Roselien Steur. 2011. *Sketching: The Basics*. Bis Publishers.

Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. 2016. Fidelity vs. simplicity: a global approach to line drawing vectorization. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 120.

Yotam Gingold, Takeo Igarashi, and Denis Zorin. 2009. Structured annotations for 2D-to-3D modeling. In *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, Vol. 28.

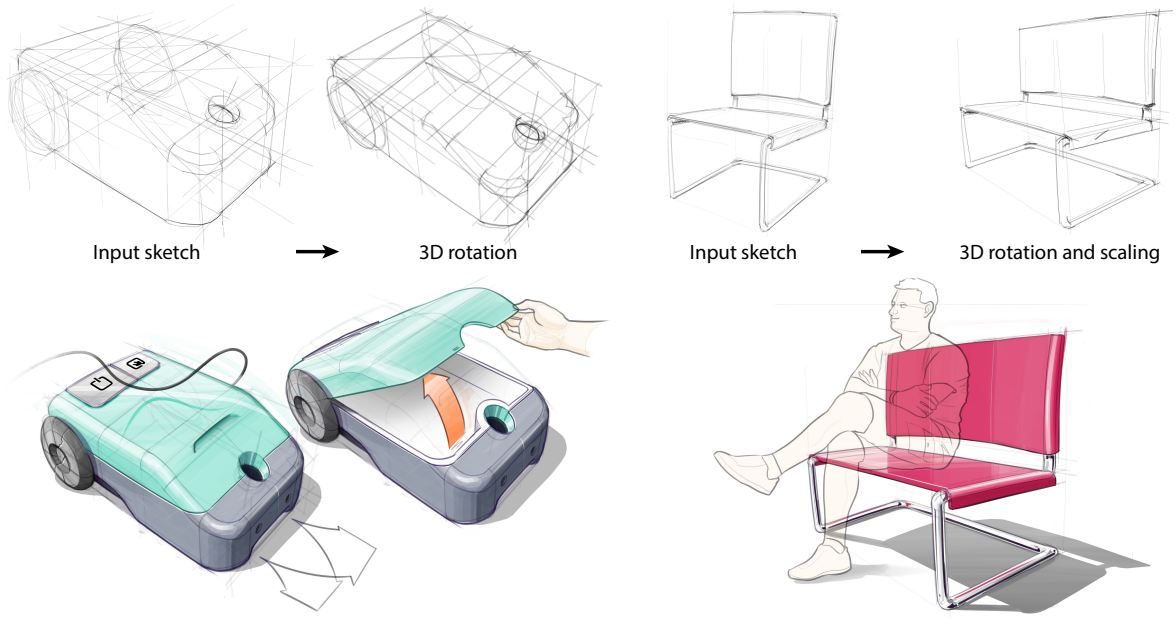


Fig. 14. **Application scenarios.** Designers can use our reconstruction algorithm to adjust the 3D pose and proportions of the drawn shape, and use the result as an underlay to create a polished presentation drawing. Here a view from the top is adopted to better show the opening of the vacuum cleaner (left), while 3D-aware scaling is employed to turn the chair into a bench (right). The presentation drawings were created by ©Robert Smit (www.flatland.agency).

Yulia Gryaditskaya, Mark Sypesteyn, Jan Willem Hoftijzer, Sylvia Pont, Fredo Durand, and Adrien Bousseau. 2019. OpenSketch: A Richly-Annotated Dataset of Product Design Sketches. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* (2019).

Erwan Guillou, Daniel Meneveaux, Eric Maisel, and Kadi Bouatouch. 2000. Using vanishing points for camera calibration and coarse 3D reconstruction from a single image. *The Visual Computer* 16, 7 (2000).

Varsha Hedau, Derek Hoiem, and David Forsyth. 2009. Recovering the spatial layout of cluttered rooms. In *Proc. ICCV*. IEEE.

Derek Hoiem, Alexei A. Efros, and Martial Hebert. 2005. Automatic Photo Pop-Up. In *ACM Transactions on Graphics (Proc. SIGGRAPH)*. 577–584.

Kun Huang, Yifan Wang, Zihan Zhou, Tianjiao Ding, Shenghua Gao, and Yi Ma. 2018. Learning to Parse Wireframes in Images of Man-Made Environments. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. 1999. Teddy: A Sketching Interface for 3D Freeform Design. *SIGGRAPH* (1999).

Dong Kang, Mark Masry, and Hod Lipson. 2004. Reconstruction of a 3D object from a main axis system. *AAAI Fall Symposium - Making Pen-Based Interaction Intelligent and Natural* (2004).

Changjian Li, Hao Pan, Yang Liu, Alla Sheffer, and Wenping Wang. 2017. BendSketch: Modeling Freeform Surfaces Through 2D Sketching. *ACM Trans. Graph. (Proc. SIGGRAPH)* 36, 4 (2017).

Changjian Li, Hao Pan, Yang Liu, Xin Tong, Alla Sheffer, and Wenping Wang. 2018. Robust flow-guided neural prediction for sketch-based freeform surface modeling. In *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*. ACM, 238.

H Lipson and M Shpitalni. 1996. Optimization-based reconstruction of a 3D object from a single freehand line drawing. *Computer-Aided Design* 28, 8 (1996), 651 – 663. https://doi.org/10.1016/0010-4485(95)00081-X

Chenxi Liu, Enrique Rosales, and Alla Sheffer. 2018. StrokeAggregator: consolidating raw sketches into artist-intended curve drawings. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 97.

Jianzhuang Liu, Liangliang Cao, Zhenguo Li, and Xiaoou Tang. 2008. Plane-based optimization for 3D object reconstruction from single line drawings. *IEEE Trans. Pattern Anal. Mach. Intell.* 30, 2 (2008), 315–327.

Xueting Liu, Xiangyu Mao, Xuan Yang, Linling Zhang, and Tien-Tsin Wong. 2013. Stereoscopizing Cel Animations. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 32, 6 (November 2013), 223:1–223:10.

Xueting Liu, Tien-Tsin Wong, and Pheng-Ann Heng. 2015. Closure-aware Sketch Simplification. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 34, 6 (November 2015).

Zhaoliang Lun, Matheus Gadelha, Evangelos Kalogerakis, Subhransu Maji, and Rui Wang. 2017. 3D shape reconstruction from sketches via multi-view convolutional networks. In *IEEE International Conference on 3D Vision (3DV)*. 67–77.

Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. 2007. FiberMesh: designing freeform surfaces with 3D curves. *ACM transactions on graphics (TOG)* 26, 3 (2007), 41.

Simon Niklaus, Long Mai, Jimei Yang, and Feng Liu. 2019. 3D Ken Burns Effect from a Single Image. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 38, 6, Article 184 (Nov. 2019), 15 pages.

Pauline Olivier, Renaud Chabrier, Damien Rohmer, Eric De Thoisy, and Marie-Paule Cani. 2019. Nested Explorative Maps: A new 3D canvas for conceptual design in architecture. *Computers and Graphics (Proc. SMI)* 82 (2019).

GüNay Orbay and Levent Burak Kara. 2012. Sketch-based surface design using malleable curve networks. *Comput. Graph. Forum* 36, 8 (2012).

Radu Orghidan, Joaquim Salvi, Mihaela Gordan, and Bogdan Orza. 2012. Camera calibration using two or three vanishing points. In *FedCSIS*.

Patrick Paczkowski, Min H. Kim, Yann Morvan, Julie Dorsey, Holly Rushmeier, and Carol O'Sullivan. 2011. Insitu: Sketching Architectural Designs in Context. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 30, 6 (2011).

Scott Robertson and Thomas Bertling. 2013. *How to Draw: drawing and sketching objects and environments from your imagination.* Design Studio Press.

Ryan Schmidt, Azam Khan, Gord Kurtenbach, and Karan Singh. 2009a. On Expert Performance in 3D Curve-Drawing Tasks. In *Proc. Sketch-Based Interfaces and Modeling*.

Ryan Schmidt, Azam Khan, Karan Singh, and Gord Kurtenbach. 2009b. Analytic drawing of 3D scaffolds. In *ACM transactions on graphics (TOG)*, Vol. 28. ACM, 149.

Cloud Shao, Adrien Bousseau, Alla Sheffer, and Karan Singh. 2011. CrossShade: shading concept sketches using cross-section curves. *ACM Transactions on Graphics* 31, 4 (2011).

Tianjia Shao, Wilmot Li, Kun Zhou, Weiwei Xu, Baining Guo, and Niloy J. Mitra. 2013. Interpreting Concept Sketches. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 32, 4 (2013), 10.

Alex Shtof, Alexander Agathos, Yotam Gingold, Ariel Shamir, and Daniel Cohen-Or. 2013. Geosemantic Snapping for Sketch-Based Modeling. *Computer Graphics Forum* 32, 2 (2013), 245–253.

Edgar Simo-Serra, Satoshi Iizuka, Kazuma Sasaki, and Hiroshi Ishikawa. 2016. Learning to Simplify: Fully Convolutional Networks for Rough Sketch Cleanup. *ACM Trans. Graph. (Proc. SIGGRAPH)* 35, 4 (2016).

Daniel Sýkora, Ladislav Kavan, Martin Čadík, Ondřej Jamriška, Alec Jacobson, Brian Whited, Maryann Simmons, and Olga Sorkine-Hornung. 2014. Ink-and-Ray: Bas-Relief Meshes for Adding Global Illumination Effects to Hand-Drawn Characters.

ACM Transaction on Graphics 33, 2 (2014), 16.

Chao Tian, Mark A. Masry, and Hod Lipson. 2009. Physical sketching: Reconstruction and analysis of 3D objects from freehand sketches. *Computer Aided Dessign* 41 (2009), 147–158.

Anil Usumezbas, Ricardo Fabbri, and Benjamin B. Kimia. 2016. From multiview image curves to 3D drawings. In *Proc. European Conference on Computer Vision (ECCV)*.

Yingze Wang, Yu Chen, Jianzhuang Liu, and Xiaoou Tang. 2009. 3D reconstruction of curved objects from single 2D line drawings. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Baoxuan Xu, William Chang, Alla Sheffer, Adrien Bousseau, James McCrae, and Karan Singh. 2014. True2Form: 3D curve networks from 2D sketches via selective regularization. *ACM Transactions on Graphics* 33, 4 (2014).

Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. 1996. SKETCH: An Interface for Sketching 3D Scenes. *SIGGRAPH* (1996).

Yichao Zhou, Haozhi Qi, Yuexiang Zhai, Qi Sun, Zhili Chen, Li-Yi Wei, and Yi Ma. 2019. Learning to Reconstruct 3D Manhattan Wireframes From a Single Image. In *Proc. IEEE International Conference on Computer Vision (ICCV)*.

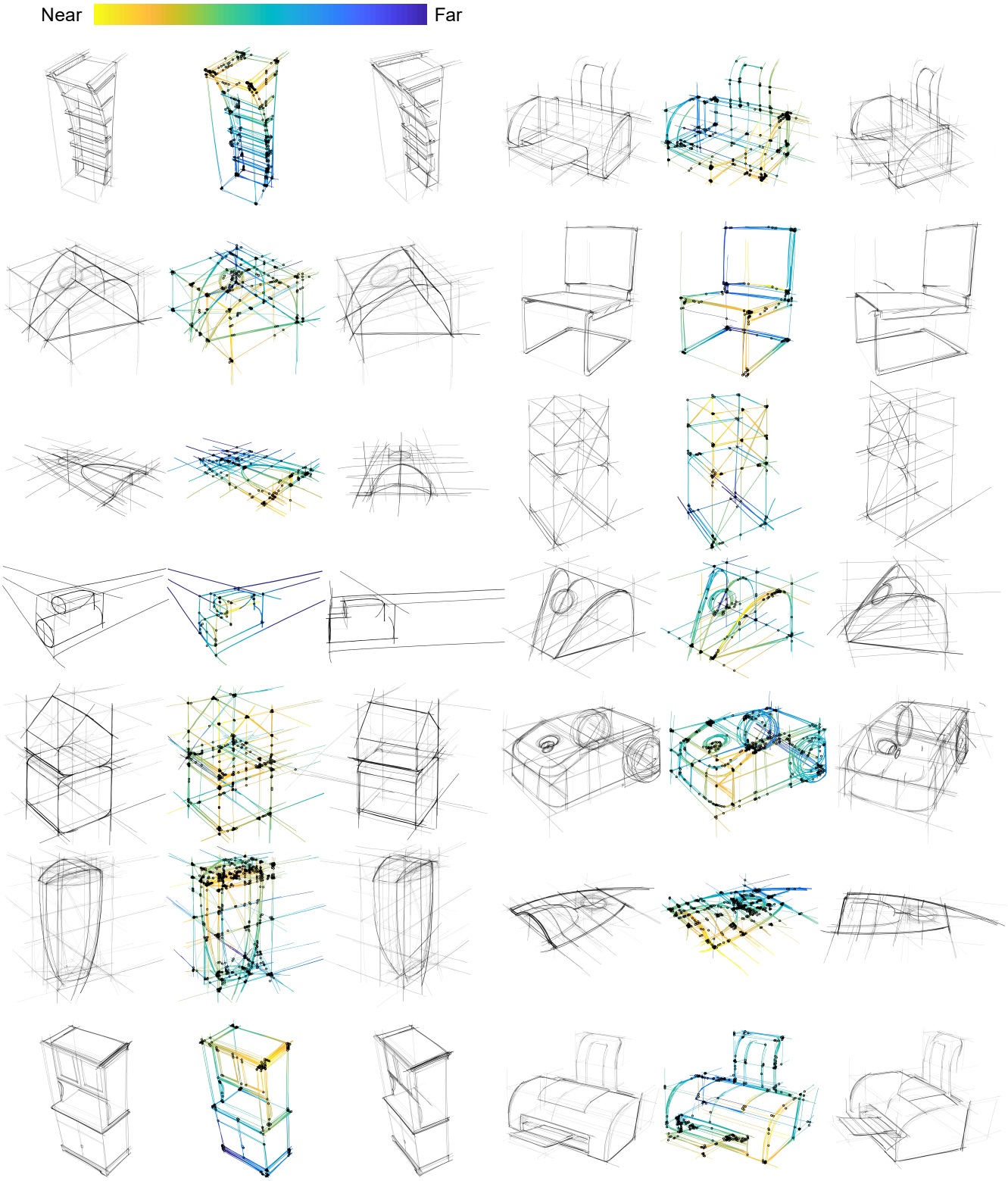


Fig. 15. **Our 3D reconstructions on a variety of industrial design sketches.** For each result, we provide the input sketch (left), a visualization of the recovered depth and 3D intersections (middle), and an alternative viewpoint (right).